בס"ד

ליאור פרץ

ראובן קליין

דו"ח מיני פרויקט 1 – שיפור תמונה

השיפור הנבחר: Glossy Surfaces and Diffused Glass

1. Glossy Surfaces

הבעיה:

השתקפות על פני כל המשטחים נראית כמו השתקפות במראה. במציאות ההשתקפות על פני משטחים שונים נראית מעט מרוחה ומטושטשת מאחר וקני ההשתקפות מתפזרות בגלל המבנה המולקולרי של החומר.

הסיבה התכנותית לבעיה:

עבור השתקפות אנחנו שולחים רק קרן השתקפות אחת כדי לחשב את הצבע המתקבל ממנה.

פתרון:

עבור כל קרן השתקפות נגדיר משטח מטרה ריבועי שאליו הקרן מאונכת. נחלק את המשטח לריבועים, ונשלח קרניים אל עבר נקודה רנדומלית בכל אחד מן הריבועים. נקבע שהצבע של נקודת המוצא יהיה הממוצע של הצבע המתקבל מכל הקרניים.

יישום הפתרון:

למחלקה Material הוספנו משתנה בשם kGlossy המבטא את רמת הברק של החומר (100 משמעותו 100% מבריק, דהיינו מראה מושלמת, ו-0 משמעותו חומר כמעט לא מבריק). המשתנה הנ"ל קובע את טווח פיזור הקרניים במשטח המטרה (כאשר הפיזור הוא -100 kGlossy), מה שלמעשה בה לידי ביטוי באורך הצלע של המשטח.

במחלקה BasicRayTracer הוספנו פונקציה בשם calcColorFromBeamOfRays שמטרתה ליצור את קרניים אל עבר נקודות רנדומליות במשטח המטרה (כמות הקרניים על פי הערך של המשתנה AMOUNT_OF_RAYS), כאשר כל נקודה תחומה בריבוע אחד מתוך Grid שמחלקת את המשטח לריבועים בגודל שווה. עבור כל קרן, הפונקציה מזמנת את הפונקציה הרקורסיבית CalcColorEffect. את הצבעים סוכמים ולבסוף משתמשים בפונקציה Color.reduce כדי לחשב את הצבע הממוצע שאותו מחזירים.

(בפונקציה נעשית בדיקה עבור כל אחת מהקרניים האם היא עוברת את המשטח לצד השני. במידה וכן, היא לא נכללת בחישוב הצבע).

לצורך חישוב הוקטורים right ו-up שמגדירים את כיוון משטח המטרה, הוספנו שדה upVector במחלקה RayTracerBase שמקבל את ערך השדה vUp של המחלקה Camera.

בפונקציה calcColorFromBeamOfRays הוקטור right מוגדר כמכפלה הוקטורית upVector X rVector (כאשר rVector הוא וקטור ההשתקפות או הוקטור החודר במשטחים שקופים), והוקטור up מוגדר כמכפלה הוקטורית right X rVector.

לפונקציה calcColorFromBeamOfRays קוראים מתוך הפונקציה CalcColorEffect.

קוד הפונקציה calcColorFromBeamOfRays:

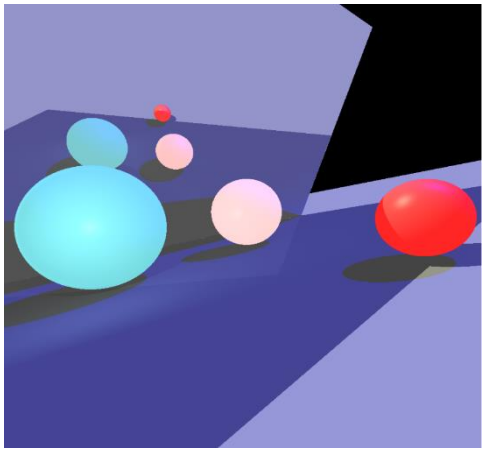
```java
/**
 * calculating recursively the global color effect of geoPoint
 * from beam of rays that scatter around a central ray.
 *
 * @param r              the central ray
 * @param n              normal to the geoPoint
 * @param level          recursion depth
 * @param kx             either transparency or reflection coefficient of the current calculated geometry
 * @param kkx            either transparency or reflection coefficient from the last recursion level
 * @param kGlossyOrClear coefficient of glossy or clear
 * @return average color from the beam of the rays
 */
private Color calcColorFromBeamOfRays(Ray r, Vector n, int level, double kx, double kkx, double kGlossyOrClear) {

    if (kGlossyOrClear == 100) {//if kGlossy is 100 the surface is perfect mirror and
                                //if KClear is 100 the surface is perfect transparent
        return calcGlobalEffect(r, level, kx, kkx);
    }
    double scatteringWidth = 100 - kGlossyOrClear;//scatteringWidth determines the edge's length of the
                                                  // target surface the rays are sent to.
                                                  // the more glossy and clear the material is, the rays less scatters.
   //building a target surface to which rays will be sent
    Point3D p0 = r.getP0();
    Vector rVector = r.getDir();
    //calculate the directions vectors of the target surface
    Vector right = rVector.crossProduct(_upVector.normalize());
    Vector up = right.crossProduct(rVector).normalize();
    rVector = rVector.scale(100);//Set the target surface at a distance of 100 from the starting point

    //get from p0 to the up left vertex of the target surface
    Point3D center = p0.add(rVector);
    Point3D upLeftCorner = center.add(up.scale(scatteringWidth).add(right.scale(-scatteringWidth)));

    //calculate the actual amount of rays (must have an integer square root)
    int squaresPerEdge = (int) Math.sqrt(AMOUNT_OF_RAYS);
    int sumOfRays = squaresPerEdge * squaresPerEdge;
    //length of each square in the grid of the target plane
     double squareLength = (scatteringWidth * 2) / squaresPerEdge;

     Vector down = up.scale(-1);
     Color color = Color.BLACK;
     //divide the target surface to squared grid
     //and send random ray to each square in it
     for (int i = 0; i < squaresPerEdge; i++) {
         for (int j = 0; j < squaresPerEdge; j++) {
             //scaling the right and down vectors in random value
             //in order to reach to a random point at the square
             //and create ray from p0 to it
             double randomRightToScale = ThreadLocalRandom.current().nextDouble(0, squareLength);
             double randomDownToScale = ThreadLocalRandom.current().nextDouble(0, squareLength);
             Vector randomVector = right.scale(randomRightToScale + j * squareLength).
                     add(down.scale(randomDownToScale + i * squareLength));
             Point3D randomPoint = upLeftCorner.add(randomVector);
             Vector randomRayDir = randomPoint.subtract(p0);

             //if the ray does not pass the surface to the other side
             // calculate the global color effect from it, else ignore it.
             if (alignZero(rVector.dotProduct(n)) * alignZero(randomRayDir.dotProduct(n)) > 0) {
                 Ray randomRay = new Ray(p0, randomRayDir);
                 color = color.add(calcGlobalEffect(randomRay, level, kx, kkx));
             } else {
                 sumOfRays--;
             }
         }
     }
     return color.reduce(sumOfRays);//average color from all rays
 }
```

הקריאה ל- calcColorFromBeamOfRays מתוך CalcColorEffect:

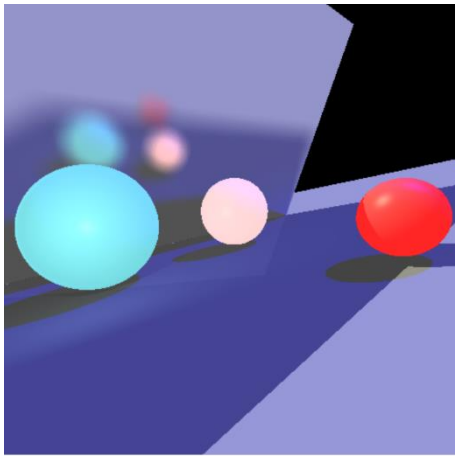
```java
double kkr = k * material._Kr;
if (kkr > MIN_CALC_COLOR_K) {
    Ray r = constructReflectedRay(gp._point, v, n);
    color = calcColorFromBeamOfRays(r, n, level, material._Kr, kkr, material._kGlossy);
}
```

דוגמאות:

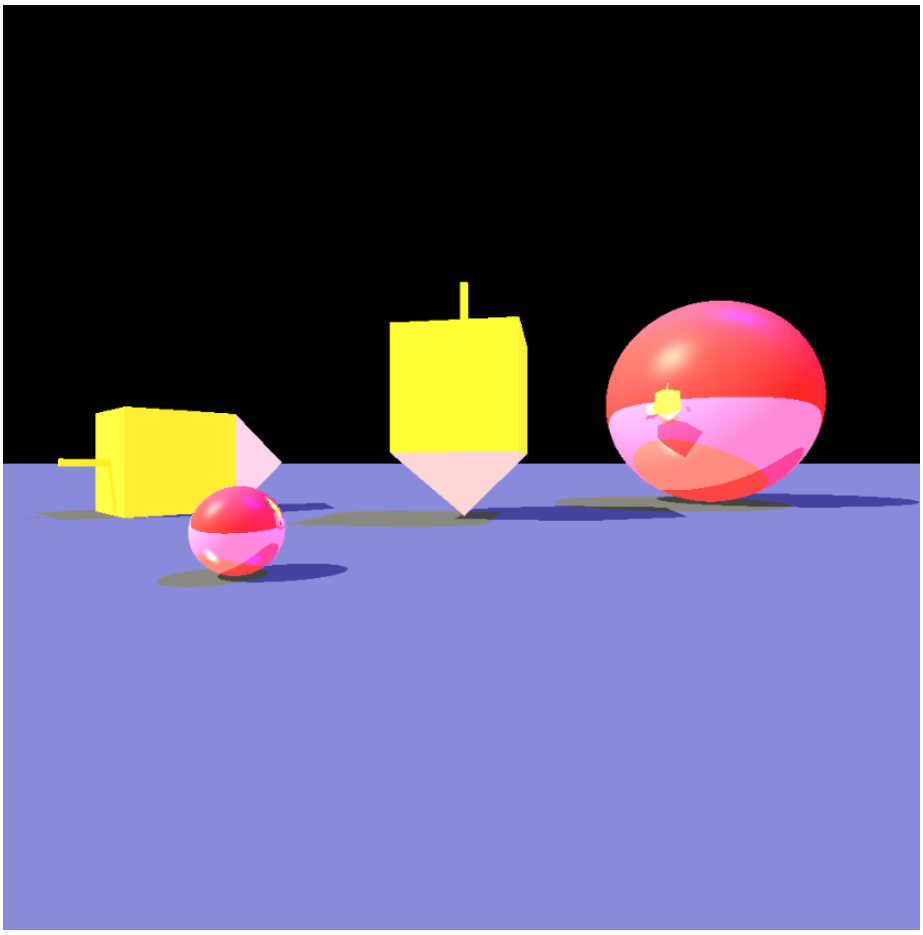
לפני-



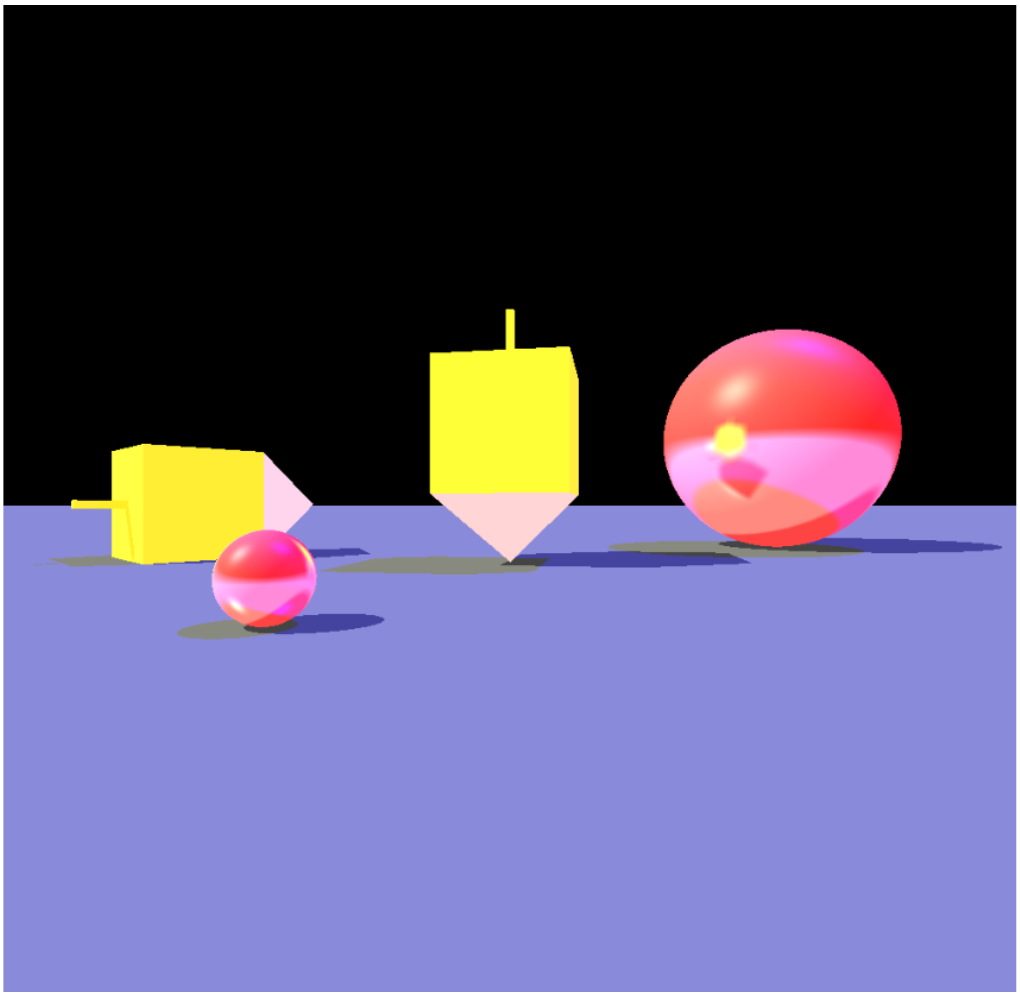
אחרי-



לפני (זו התמונה של המיני פרוייקט)-



אחרי (השתקפות הסביבון על פני הכדור)-



2. Diffused glass

<u>הבעיה:</u>
העצמים מבעד לגאומטריות המוגדרות כשקופות (kT גדול) תמיד נראים ברורים, בעוד שבמציאות יש חומרים שהם שקופים חלקית (כמו חלון של אמבטיה).

<u>הסיבה התכנותית לבעיה:</u>
מבעד לעצם שקוף אנחנו שולחים רק קרן אחת המחזירה את הצבע של מה שנמצא מולה, כך שהתמונה נראית בבירור על פני העצם השקוף.

<u>פתרון:</u>
בדומה ל-glossy surfaces, במקום לשלוח רק קרן אחת, נשלח קרניים בפיזור מסויים כלפי משטח מטרה, ונגדיר שהצבע של הנקודה הוא הממוצע של הצבע המחושב מכל הקרניים.

יישום הפתרון:

למחלקה Material הוספנו משתנה בשם kClear המבטא את רמת בהירות השקיפות של החומר (100 משמעותו שעצמים יראו דרכו בבירור, ו-0 משמעותו שעצמים יראו דרכו בצורה מטושטשת מאוד). המשתנה הנ"ל קובע את טווח פיזור הקרניים במשטח המטרה (כאשר הפיזור הוא 100-kClear), מה שלמעשה בה לידי ביטוי באורך הצלע של המשטח.

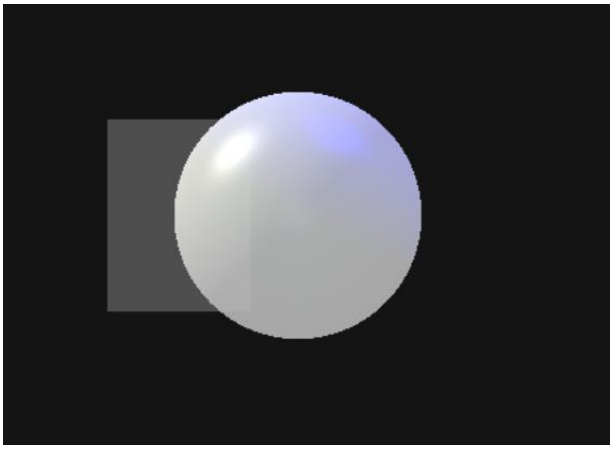
גם כאן המימוש במחלקה BasicRayTracer משתמש בפונקציה calcColorFromBeamOfRays כדלעיל.

הקריאה ל- calcColorFromBeamOfRays מתוך CalcColorEffect:

```
double kkt = k * material._Kt;
if (kkt > MIN_CALC_COLOR_K) {
    Ray r = constructRefractedRay(gp._point, v, n);
    color = color.add(
            calcColorFromBeamOfRays(r, n, level, material._Kt, kkt, material._kClear));
}
```

דוגמאות:

לפני-



אחרי-